Expérimentation d'une plate-forme de portage logiciel à l'INRIA

Yann Genevois
Service d'Expérimentation et de Développement (SED)
INRIA Rhône-Alpes
655 avenue de l'Europe
38 334 Saint Ismier

Maurice Brémond
Service d'Expérimentation et de Développement (SED)
INRIA Rhône-Alpes

Nicolas Turro
Service d'Expérimentation et de Développement (SED)
INRIA Rhône-Alpes

Roger Pissard
Service d'Expérimentation et de Développement (SED)
INRIA Rhône-Alpes

Eric Ragusi
Service d'Expérimentation et de Développement (SED)
INRIA Rhône-Alpes

Résumé

Au sein de l'INRIA, les équipes projets qui développent des logiciels scientifiques sont confrontées, dès lors qu'elles veulent les distribuer, au portage sur une multiplicité d'environnements matériels et logiciels en constante évolution. Les tâches de portage logiciel sont réalisées par l'intermédiaire de leurs propres parcs de machines hétérogènes qui posent des problèmes d'hébergement, d'administration système et de maintenance.

L'installation préalable de logiciels et librairies spécifiques, la validation de l'installation, ou encore le développement de composants systèmes nécessitent des interventions administrateurs. Ces configurations spécifiques sur de nombreuses combinaisons possibles d'architectures et de systèmes ne permettent pas d'envisager une solution de mutualisation à partir de serveurs de connexion classiques.

PIPOL (Plate-forme Inria de POrtage Logiciel) est une expérimentation d'un service de portage pour les logiciels scientifiques à l'échelle de l'INRIA. Les ressources matérielles sont partagées et disponibles après réservation et déploiement du système d'exploitation souhaité. Un service permet la configuration automatique des systèmes, la planification et le traitement de travaux utilisateurs.

La viabilité de ce service, liée au coût de son administration ainsi qu'aux services rendus à la communauté, est illustrée par des retours utilisateurs.

Mots clefs

développement logiciel, portage logiciel, compilation, informatique à la demande, réservation de ressources, déploiement dynamique de systèmes

1 Contexte et problématique

L'INRIA, Institut National de Recherche en Informatique et en Automatique, a pour vocation d'entreprendre des recherches fondamentales et appliquées dans les domaines des sciences et technologies de l'information et de la communication. L'institut assure également un transfert de technologies en accordant une grande attention à la formation par la recherche, à la diffusion

de l'information scientifique et technique, au développement, à l'expertise et à la participation à des programmes internationaux. Les équipes projets de l'institut qui développent des logiciels scientifiques sont confrontées, dès lors qu'elles veulent les distribuer, au portage sur une multitude d'environnements matériels et logiciels en constante évolution. Les équipes de développement réalisent les portages logiciels par l'intermédiaire de leurs propres parcs de machines hétérogènes qui posent des problèmes d'hébergement, d'administration système et de maintenance. Pour être plus précis, on définit le portage comme l'ensemble des tâches nécessaires pour faire fonctionner un logiciel sur une architecture, un système d'exploitation, une distribution ou une version de librairies différents de l'environnement initialement utilisé pour sa conception. Ceci inclut l'adaptation, l'édition et la compilation du code ainsi que les tests et mise en paquetage du logiciel.
C'est pourquoi, depuis 2006, la D2T (Direction du Développement Technologique) pilote une expérimentation d'un service centralisé à l'échelle de l'INRIA: PIPOL (Plate-forme Inria de POrtage Logicielle) destinée au portage des logiciels scientifiques sur les architectures et systèmes utiles pour leur diffusion. La résolution de cette problématique nécessite la mise à disposition de ressources (architectures, systèmes, logiciels...) auprès des infrastructures de recherche avec les contraintes que cela implique (disponibilité, sécurité...).

2 Spécification d'une solution

Le portage logiciel nécessite l'utilisation d'outils spécifiques (compilations, tests, paquetages). La configuration de l'environnement utilisateur pour les travaux de portage nécessite des interventions avec accès administrateurs pour, par exemple, l'installation préalable de logiciels et librairies, la validation de l'installation sur un nouveau système, ou le développement de nouveaux composants systèmes. Dans le cadre d'un service centralisé pour plusieurs projets, les nombreuses combinaisons possibles d'architectures et de systèmes avec configurations spécifiques et la mutualisation des ressources ne permettent pas d'envisager des machines statiques pré-installées comme pour la « ferme de compilation gcc »[1] et la « ferme de compilation PostgreSQL »[2] par exemple. Pour que la plate-forme soit attractive, son utilisation doit être aisée, le temps d'installation des systèmes demandés doit rester raisonnable et des services complémentaires doivent être proposés pour accompagner le développement logiciel. La viabilité d'une telle plate-forme est liée au coût de son administration ainsi qu'à la variété des systèmes proposés.
La plate-forme de portage doit permettre aux projets de recherche de disposer de différentes architectures matérielles et logicielles (OS, librairies, outils de développement...) pour valider la portabilité de leurs logiciels et cela en respectant plusieurs critères:

- **la plate-forme doit être extensible.** De nouveaux systèmes, architectures ou distributions logicielles doivent pouvoir y être intégrés, à la demande des utilisateurs. Les fréquentes mises à jours des systèmes (changement de version tous les six mois pour certaines distributions Linux) doivent être prises en compte ;

- **l'utilisateur doit avoir un accès administrateur** sur les systèmes pour rendre possible la personnalisation de son environnement, l'installation de librairies ou logiciels sans intervention administrateur. Les accès aux ressources de la plate-forme doivent être actifs pour un temps déterminé, correspondant à la durée des travaux de portage ;

- **la maintenance doit être simple.** La plate-forme doit être administrable à moindre coût. La prise en main à et la gestion de commandes à distance, sans présence physique nécessaire, sont des techniques facilitant le maintient d'une plate-forme. La contrainte d'accès privilégié laisse l'utilisateur totalement maître de son environnement. La gestion des machines ne peut donc pas être basée sur le système déployé mais doit s'orienter vers des commandes physiques indépendantes ;

- **une facilité d'utilisation.** Les utilisateurs doivent être guidés à travers différentes documentations sur la prise en main de la plate-forme, l'utilisation des systèmes disponibles et le jeu de commandes utiles dont ils disposent. Pour rendre plus claires les fonctionnalités disponibles et la présentation des données aux utilisateurs, une interface peut être utilisée.

3 Mise en œuvre de la plate-forme

PIPOL est une expérimentation qui porte sur la faisabilité, la viabilité et le dimensionnement matériel d'un service de portage. À l'heure actuelle le service n'est ouvert qu'à un nombre restreint d'utilisateurs, mais suffisant pour évaluer le coût humain et le dimensionnement matériel que nécessiterait son passage à une plus grande échelle.

3.1 Principe de fonctionnement

On distingue trois étapes pour l'obtention d'une session sur un système PIPOL (Cf. figure 1):

- le choix et la réservation des ressources ;
- le déploiement d'un système correspondant aux ressources choisies ;
- la configuration dynamique du système déployé.

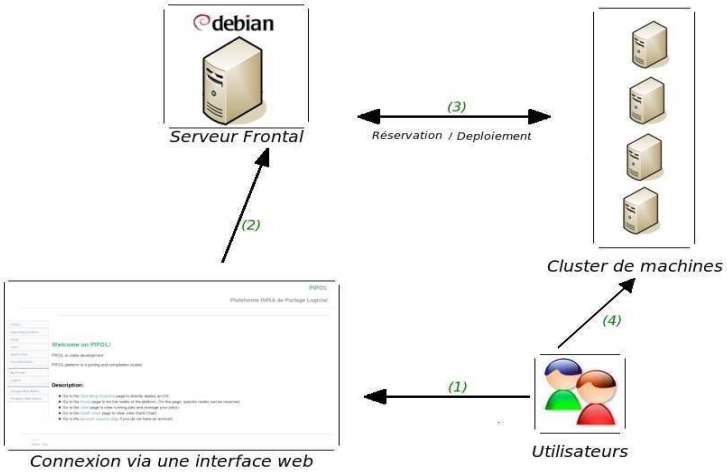

Figure 1: Principe de fonctionnement de la plate-forme

Les ressources réservables sur la plate-forme sont de plusieurs types distincts selon qu'il est question de matériel physique ou de système d'exploitation. Dans notre cas, les ressources sont définies comme :

- une architecture physique: Intel 32 bits ou 64 bits, Mac Intel, Itanium ;
- une architecture virtuelle : Xen 32 ou 64 bits ;
- une licence pour les systèmes qui le demandent : Windows, Mac Os X, Suse, RedHat, Mandriva...

La connexion sur un système à licence nécessite donc la disponibilité de deux ressources : une machine physique ou virtuelle et une licence. L'utilisateur peut obtenir l'usage de ressources pour un temps donné, dès qu'elles sont disponibles ou à une date choisie.
La plate-forme PIPOL est basée sur un déploiement dynamique, après réservation, d'un système d'exploitation. Durant la phase de déploiement, une image de référence du système à déployer est recopiée depuis un espace de stockage vers l'architecture qui veut l'exécuter (machine physique ou virtuelle). Cette mécanique doit être indépendante du système à déployer pour une plus grande interopérabilité et une plus grande facilité d'ajout de systèmes/architectures.
Une fois l'image copiée sur l'architecture réservée, le déploiement se termine par l'exécution des commandes de personnalisation. L'accès administrateur configuré en fin de déploiement permet l'installation et la configuration logicielle nécessaire aux tests de portage. L'espace de stockage personnel sur la machine frontale est aussi rendu accessible sur le système déployé pour exécuter des scripts automatiques de configuration réalisés par les utilisateurs.

3.2 Mise en œuvre

La plate-forme PIPOL se compose d'un serveur frontal (un Dell 2950), d'un serveur de virtualisation (Dell 1950) et d'un cluster de machines hétérogènes (Dell 1950, Mac Intel, Itanium). Un réseau Gigabit/s relie les différents nœuds au serveur frontal. Les répertoires utilisateurs et les images sont stockés sur des espaces RAID de 1To. Les machines physiques sont contrôlées par des cartes de management. Ces cartes offrent l'accès à une console distante quelque soit l'état du réseau de la machine réservée pour s'affranchir des connexions SSH ou permettre à l'utilisateur de travailler sur des composants système de bas niveau (modules noyaux par exemple). Les machines virtuelles sont contrôlées par un serveur de virtualisation XEN [3].

La réservation des ressources est orchestrée par OAR[4], logiciel développé par l'équipe MESCAL de l'INRIA (notamment utilisé dans la grille Grid'5000[5]). OAR, grâce à un ordonnanceur périodique de tâches peut réorganiser dynamiquement les

réservations afin d'optimiser l'utilisation des ressources. Les réservations peuvent être définies avec différentes priorités grâce à la gestion de files d'attente OAR. Une série de règles d'admissions, comme la limitation du nombre de ressources réservables en une fois ou encore la gestion de licences, valide la demande de réservation de ces ressources. Ces règles, personnalisables, définissent le fonctionnement et les limites des réservations. Toutes les ressources pouvant être réservées doivent être définies dans des bases de données Postgresql[6] conjointement aux données nécessaires au fonctionnement d'OAR. Ces informations sont accessibles par des lignes de commandes sur le serveur frontal ou encore via l'interface web de PIPOL. L'exécution de deux scripts en début et en fin de réservation par le système OAR permet de définir des tâches à accomplir à chaque demande d'accès aux ressources ; dans le cas de PIPOL, le déploiement d'un système d'exploitation et la gestion des comptes utilisateur sur les cartes de management.

Pour le déploiement des images, des systèmes déjà existants ont été envisagés (comme Kadeploy[7], outil réalisé pour Grid'5000 ou des solutions Open Source comme SystemImager[8] ou G4u[9] qui ne prennent pas en charge toutes les architectures). Les contraintes d'utilisation (diversité des systèmes et des architectures matérielles) nous ont conduit à développer notre propre système. L'installation des images est effectué avec un noyau de déploiement pour les machines physiques et par des images LVM pour la virtualisation. Dans le cas d'une machine physique, le déploiement est réalisé à partir d'un système personnalisé comprenant un noyau Linux et un système de fichiers chargé en RAM grâce au protocole PXE (démarrage par le réseau). Le noyau et le système de fichiers sont récupérés par TFTP au démarrage de la machine, puis l'initrd du noyau de déploiement charge un script présent sur le serveur par TFTP pour l'exécuter. Dans le cadre d'un déploiement, le script attend l'arrivée de données sur un port pré-défini et les écrit sur le disque dur local. Le serveur frontal quant à lui, se charge de transmettre l'image système par le réseau et via une commande netcat en direction du nœud. Les systèmes d'exploitation sont stockés sur le serveur en tant qu'images systèmes au format « dd » compressées avec gzip. La compression des images offre deux avantages par rapport à une transmission d'images brutes: la diminution de l'espace disque nécessaire pour stocker les images et un temps de transfert diminué (en moyenne 10 minutes pour charger un système). On observe des différences de fonctionnement entre les architectures Mac Intel, PC et Itanium qui nécessitent des adaptations. Les nœuds de type PC peuvent être configurés de façon à démarrer sur un autre périphérique (disque dur local) si le démarrage par le réseau échoue alors que le démarrage des machines Itaniums ne peut définir qu'un seul périphérique de démarrage. Le démarrage des Itaniums se fait donc uniquement par le réseau, ce qui nécessite d'extraire les noyaux des images pour les faire démarrer sur le disque dur. Ce procédé de déploiement doit donc être adapté en fonction de chaque nouvelle architecture, même si le principe reste identique.

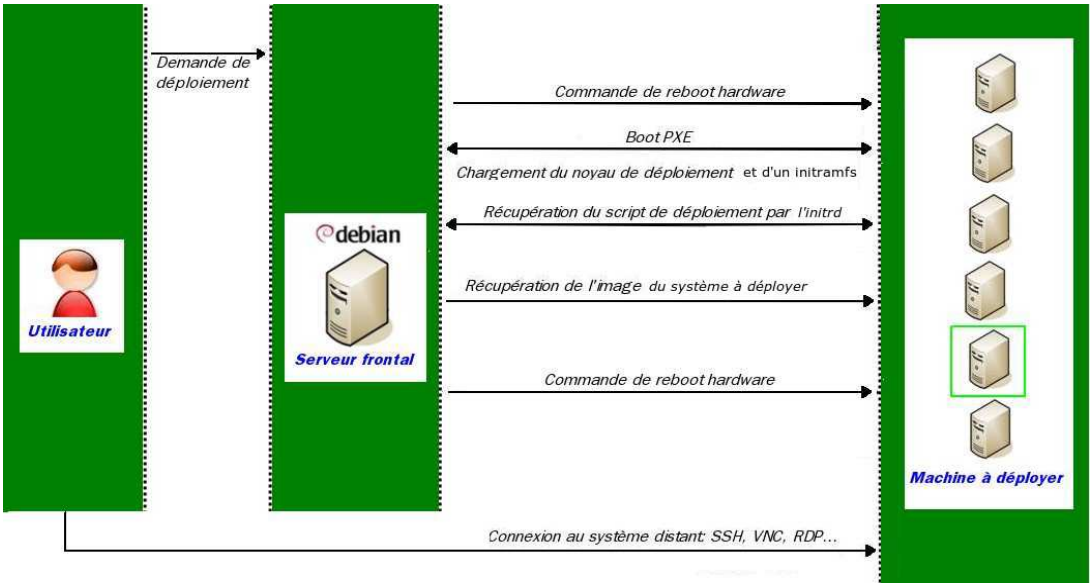

Figure 2: Principe de déploiement d'une image

Le déploiement d'un système virtualisé à l'aide de Xen est basé sur des images LVM en lecture seule. Les images virtualisées sont stockées sur un serveur dédié. Lors d'un déploiement, un snapshot est attaché aux images permettant de stocker les modifications apportées par l'utilisateur sans les altérer. Plusieurs machines virtuelles peuvent ainsi être exécutées en utilisant la même image de référence.

Un script de post-installation, dépendant de l'architecture et du système, est exécuté afin de configurer les accès utilisateurs : export par NFS du répertoire utilisateur et de logiciels partagés, configuration des accès SSH, configuration des accès VNC ou RDESKTOP, configuration de variables d'environnement et configuration de la commande sudo sans mot de passe afin de la

rendre scriptable. Cette personnalisation des images dépend du système déployé. Le système de post-installation mettant en place ces services sur les nœuds se base sur un système modulaire permettant l'exécution de scripts dont la nomenclature correspond à celle des images. Le déploiement se termine par des RunCommands utilisateurs (RC) exécutés sur le principe de filtrage entre leurs noms et celui de l'image.

4 Les fonctionnalités de haut niveau

4.1 Les automatisations

L'utilisateur dispose de trois niveaux d'automatisations :

– **l'automatisation de la personnalisation de son système**. La fin de la configuration du système déployé comprend l'exploitation d'une série d'exécutables réalisés par les utilisateurs. Ces fichiers sont des scripts ou des binaires localisés dans l'espace personnel du serveur. Leurs noms commencent par « **rc** » (pour « run command' ») et peuvent être suffixés par un élément caractéristique de la nomenclature des images pour permettre le filtrage par type de système. Ils sont indépendants des images et sont modifiables sur le serveur frontal sans qu'un déploiement ne soit réalisé. Exécutés avec les droits administrateurs sur le système déployé, leur vocation principale est l'installation de paquets spécifiques depuis des dépôts externes ;

– **l'exécution automatique, après déploiement, d'un script ou d'un binaire.** À la manière de qsub (PBS[10]) ou de oarsub (OAR), une commande de soumission de travaux est disponible: **pipol-sub.** Elle permet de faire exécuter une commande sur plusieurs systèmes après déploiement. À la différence des scripts « run command » de post-installation, l'exécution est faite sous l'identité de l'utilisateur, le travail n'étant plus de configurer le système. Typiquement, la commande utilisateur peut être un script qui récupère des sources depuis un gestionnaire de version externe (comme la gforge[11]), puis effectue une compilation et des tests. Les résultats sont consultables dans l'espace utilisateur. Ils peuvent aussi être synthétisés avec un service de type Cdash[12]. Une fonctionnalité similaire à BuildBot[13] est expérimentée. **pipol-sub** permet de recopier directement un répertoire de sources dans l'image déployée sans avoir à utiliser un gestionnaire de versions. À la différence de ce qui est fait dans BuildBot, l' image déployée ne conservant pas d'historique, il ne s'agit pas d' une synchronisation incrémentale ;

– **l'exécution de travaux cycliques.** Les scripts présents dans un espace utilisateur spécifique sont exécutés la nuit avec une priorité.

4.2 L'interface Web

Il s'agit d'une couche logicielle s'appuyant sur les outils décrits précédemment: OAR et pipol-sub, et développée à partir du « framework » Modèle-Vue-Contrôleur Django[14]. La simplicité de prise en main de cet outil pouvant interagir avec des bases de données et l'utilisation du langage Python, de par l'expertise des administrateurs de la plate-forme, ont orienté le choix du développement Web vers ce logiciel. Django est un framework en Python interfaçant des bases de données (les Modèles), des pages HTML (les Vues) et un système de contrôles et de validation des données (les Contrôleurs). PIPOL intègre grâce à Django des modèles de données définissant les ressources du cluster pour fournir notamment les services de:

– **consultation des informations du cluster** : visualisation de l'état des ressources et des travaux en cours, suivie de l'avancement d'un déploiement ;

– **réservation des ressources et déploiement des systèmes ;**

– **gestion des travaux**: suppression ou prolongation d'une réservation ;

– **centralisation des documentations.** L'interface Web regroupe différentes documentations utilisateur et administrateur: manuel utilisateur, listing des logiciels et paquets contenus dans les images, aide à la prise en main sur les différents systèmes d'exploitation.

Figure 3: L'interface Web de PIPOL

4.3 L'interface d'administration

Des commandes réservées aux administrateurs de la plate-forme sont nécessaires :

– **pour tester les noyaux de déploiement :** le noyau de déploiement charge un script présent sur la plate-forme PIPOL. Les modifications de ce script peuvent être facilement testées en utilisant la commande de chargement d'un noyau ;

– **pour déployer une image sans post-configuration :** l'administration des images demande souvent de les déployer dans leur configuration d'origine pour les modifier ;

– **pour gérer les images systèmes :** les commandes de gestion des images offrent la possibilité de sauvegarder une image, de valider une image pour la mettre à disposition des utilisateurs, de revenir en arrière d'une version en cas de problèmes dans une image ou de réaliser une copie d'une image pour y faire des tests ;

– **pour la création d'une image :** automatisation de la création d'une image en fournissant certains paramètres tels que la taille de l'espace disque à sauvegarder ou le label du disque dur ;

– **pour la gestion des cartes de management :** activation/désactivation des comptes, changement des mots de passe ;

– **pour la mise à jour des systèmes.** La mise à jour des images systèmes se fait de façon automatique le week-end : les images sont déployées, mises à jour avec les dépôts en ligne et vérifiées avant validation. La mécanique des « run command » utilisateur est employée pour ce travail.

5 Les retours d'exploitation

À partir de cette expérimentation, il est important d'analyser les retours d'exploitation du côté administration et du côté utilisateur, qui permettront de définir un service en production dimensionné, utile et viable.

Retour d'administration

À partir de notre expérience, nous concluons qu'un ingénieur à plein temps est nécessaire pour gérer une telle plate-forme. Cette gestion se décompose en deux mi-temps avec deux niveaux de technicité différents :

– la maintenance et le support utilisateur : création de comptes pour les nouveaux utilisateurs et aide pour la prise en main, support technique aux utilisateurs, réponse aux questions sur les listes de diffusion, maintenance des images et du système ;

– le suivi de l'évolution de la plate-forme qui consiste, essentiellement à : ajouter des fonctionnalités à la demande de l'utilisateur pour améliorer le service (ex: accès console à distance) ou pour améliorer l'administration (ex: mise à jour automatique) et à intégrer de nouvelles architectures matérielles et logicielles.

Retour d'utilisation

Après un semestre de montée en charge de la plate-forme (cf. figure 4), un sondage auprès des utilisateurs a été réalisé durant le mois de mars 2009. A cette époque, nous avions une cinquantaine d'utilisateurs effectifs. Un quart ont répondu à ce sondage, dont le but était de récolter les types et les retours d'utilisation, regroupant une dizaine d'équipes projets sur six Centre de Recherche Inria.

La synthèse que l'on peut faire de ces retours de sondage concerne deux points :

– **au niveau du type d'utilisation**, les utilisateurs sont réguliers (hebdomadaire pour plus de la moitié) et avec des sessions de travail interactives (pour les trois-quarts). L'autre utilisation typique (un quart) est en mode automatique : lancement de « builds » ou de tests réguliers. Toutes les familles de systèmes (MacOS, Windows, Linux) proposées sont utilisées avec une préférence pour les différentes distributions Linux ;

– **au niveau des retours d'utilisations**, ils sont unanimement positifs sur l'utilité, la prise en main et l'ergonomie. Les améliorations demandées portent principalement sur un déploiement plus rapide et la mise à disposition d'architectures peu disponibles ou obsolètes.

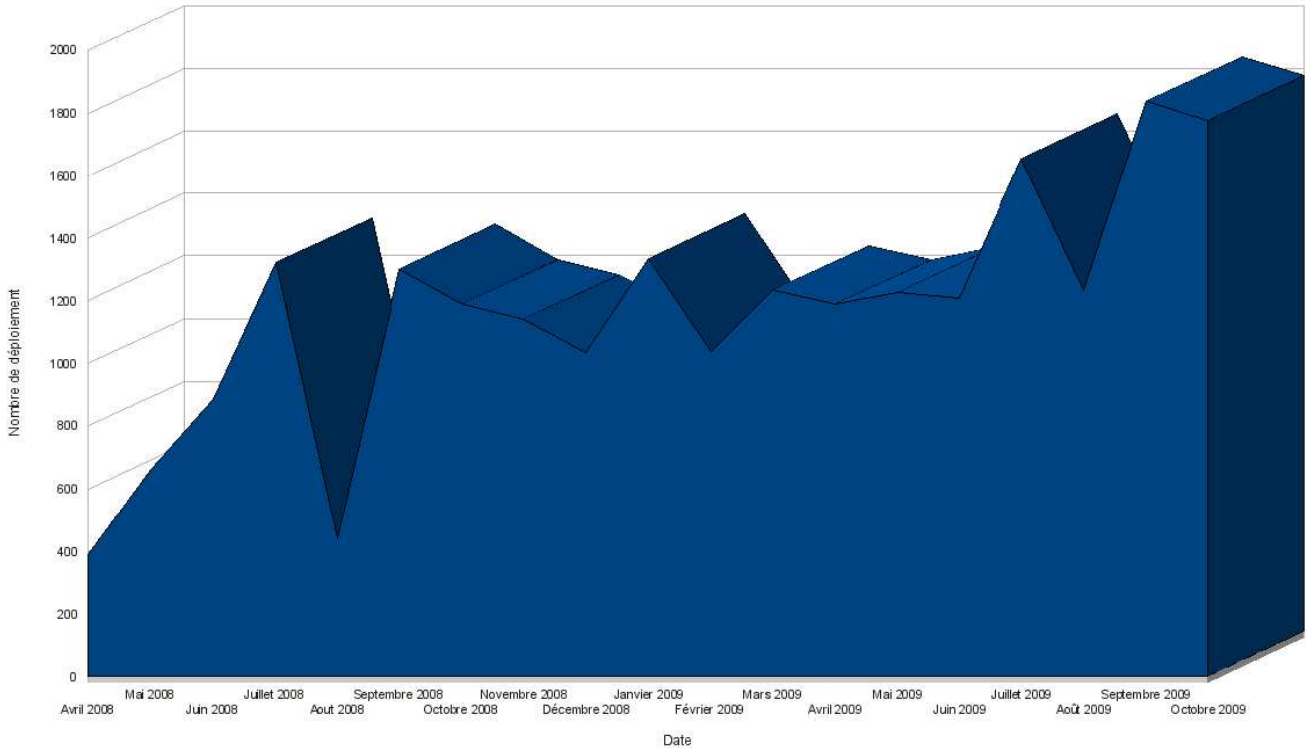

Figure 4: Montée en charge de la plate-forme depuis Avril 2008

6 Perspective et conclusion

L'expérimentation PIPOL a atteint ses objectifs : elle nous a permis de définir la faisabilité et la viabilité d'un tel service de portage mutualisé. Ainsi, dans la mesure où les ressources sont disponibles, les utilisateurs peuvent disposer en moins de dix minutes d'une architecture physique avec un système d'exploitation installé et un accès administrateur. Le succès d'utilisation valide la concordance entre le concept de la plate-forme et les besoins des utilisateurs. La plate-forme est utilisée en particulier par les équipes projets qui développent CADP, MUMPS ou MPFR.

Pour passer en production à l'INRIA, à une plus large échelle, la plate-forme est évolutive : on peut ainsi ajouter des machines en fonction de l'utilisation croissante, son concept général permet l'intégration de nouvelles architectures et de nouveaux systèmes. Si l'on veut avoir un bon niveau de service, le travail d'un ingénieur plein-temps est nécessaire pour supporter l'animation, l'évolution et l'administration de la plate-forme.
La plate-forme est opérationnelle et son utilisation à continue progresser (plus de 1800 déploiements au mois de septembre 2009) mais nous attendons des arbitrages de ressources pour effectuer ce passage à l'échelle.
Dans le futur, nous considérons les évolutions suivantes :

- sauvegarde d'images systèmes utilisateurs, avec des ressources disques supplémentaires ;
- changement du format de stockage des images systèmes pour s' affranchir de la géométrie des disques ;
- extension du service sur architectures virtuelles : les déploiements étant plus rapides, il serait également possible de proposer un service d'intégration continue ;
- réservation d'un cluster pour les tests de parallélisation.

Le passage en production de PIPOL doit permettre à la communauté des développeurs INRIA de disposer d'un outil complémentaire à l'INRIA-Gforge dans le cycle de développement. Ces outils sont indispensables pour faire adopter aux développeurs les bonnes pratiques de développement afin d'améliorer la qualité des productions logicielles.

7 Références

[1] Gcc compile farm, http://gcc.gnu.org/wiki/CompileFarm, accédé le 31 Octobre 2009

[2] PostgreSQL compile farm, http://www.pgbuildfarm.org/cgi-bin/show_status.pl, accédé le 31 Octobre 2009

[3] The Xen hypervisor, http://xen.org/, accédé le 31 Octobre 2009

[4] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Martin, Gr ▫gory Mouni ▫, Pierre Neyron, and Olivier Richard. *A batch scheduler with high level components in Cluster computing and Grid,* 2005

[5] Bolze RaphaÎl, Cappello Franck, Caron Eddy, Dayde Michel, Desprez Frédéric, Jeannot Emmanuel, Jegou Yvon, Lantéri Stephane, Leduc Julien, Melab Noredine, Mornet Guillaume, Namyst Raymond, Primet Pascale, Quetier Benjamin, Richard Olivier, Talbi El-Ghazali and Irea Touche, *Grid'5000: a large scale and highly reconfigurable experimental Grid testbed, Nov.* 2006

[6] Postgresql, http://www.postgresql.org/, accédé le 31 Octobre 2009

[7] Kadeploy, http://kadeploy.imag.fr/index.html, accédé le 31 Octobre 2009

[8] SystemImager, http://wiki.systemimager.org/index.php/Main_Page, accédé le 31 Octobre 2009

[9] G4U, http://www.feyrer.de/g4u/, accédé le 31 Octobre 2009

[10] Portable Batch System: PBS, http://www.pbsgridworks.com/, accédé le 31 Octobre 2009

[11] David Margery, Janet Bertot, Christophe Demarey, Claude Inglebert, *InriaGforge : leçons de 2 ans d'exploitation de GForge à l'INRIA,* Journées Réseaux 2007, 2007.**jres**.org/planning/pdf/145.pdf

[12] Cdash: an open source, web-based testing server, http://www.cdash.org/, accédé le 31 Octobre 2009

[13] Buildbot, http://buildbot.net/, accédé le 31 Octobre 2009

[14] Django, http://www.djangoproject.com/, accédé le 31 Octobre 2009